In the United States Patent and Trademark Office

Board of Patent Appeals and Interferences

Appeal Brief

In re the Application of:

Bryce Allen Curtis and Jimmy Ming-Der Hsu
Serial No. 09/377,629
Filed: August 19, 1999
Attorney Docket No. AT999179

METHOD, SYSTEM, AND PROGRAM FOR ACCESSING VARIABLES FROM AN OPERATING SYSTEM FOR USE BY AN APPLICATION PROGRAM

Submitted by:

Konrad, Raynes & Victor LLP
315 So. Beverly Dr., Ste. 210
Beverly Hills CA 90212
(310) 556-7983
(310) 556-7984 (fax)

TABLE OF CONTENTS

I. Real Party in Interest

The entire right, title and interest in this patent application is assigned to real party in interest International Business Machines Corporation.

II. Related Appeals and Interferences

Appellant is not aware of any other appeals or interferences which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

III. Status of the Claims

Claims 1, 3-10, 12-19, and 21-27 are pending.

The final rejection of the claims is being appealed for all pending claims 1, 3-10, 12-19, and 21-27.

Claims 2, 11, and 20 have been canceled during prosecution.

IV. Status of Amendments

A proposed after final amendment was filed by applicants on April 15, 2004 to respond to objections to the specification and proposing amendments to independent claims 1, 10, and 19. We are currently awaiting an indication of whether or not the proposed amendments will be entered.

V. Summary of the Invention

The presently claimed invention is directed to accessing variables from an operating

system. A command is received from an application program for at least one variable maintained by the operating system. It is determined whether the at least one variable is in a data object. If the at least one variable is in the data object, returning the at least one variable to the application program, and, if the at least one variable is not in the data object, the command from the application program is executed to store at least one variable maintained by the operating system in the data object accessible to the application program, wherein the application program is executing on the operating system. In particular, an operating system native command to use to retrieve the at least one variable is determined. The operating system native command is executed in response to the command from the application program to retrieve the at least one variable into a buffer. The retrieved at least one variable is stored from the buffer into the data object. The command from the application program is executed to retrieve the at least one variable from the data object for return to the application program.

For example, as described in the specification in connection with one embodiment, a command is received from an application program for at least one variable maintained by the operating system. It is determined whether the at least one variable is in a data object. See, for example, Applicants' Specification at page 12, lines 26-27, and FIG. 3a, block 402. If the at least one variable is in the data object, the at least one variable is returned to the application program. See, for example, Applicants' Specification at page 12, lines 27-29, and FIG. 3a, blocks 404-406.

If the at least one variable is not in the data object, then the command from the application program is executed to retrieve and store the at least one variable in the data object. In particular, an operating system native command to use to retrieve the at least one variable is determined. For example, claims 5, 14, and 23 describe that the operating system native

command is selected from a set of native operating system commands for different types of operating systems, wherein the application program is capable of executing on each of the different types of operating systems. For example, see Applicants' Specification, page 13, lines 2-5 and FIG. 3a, blocks 408-418; and page 14, lines 12-28. Thus, for different operating systems, the same command from an application program will cause different operating system native commands to be executed to retrieve a variable. Also, Applicants' Specification describes that a cross-platform program is capable of accessing operating system information using one of many available native operating system commands. For example, see Applicant' Specification, page 14, lines 12-28.

The operating system native command is executed in response to the command from the application program to retrieve the at least one variable into a buffer. For example, see Applicants' Specification, page 13, lines 11-13. The retrieved at least one variable is stored from the buffer into the data object. For example, see Applicants' Specification at page 13, lines 14-17, which indicates that environment variables are generated as a data stream, and this output data stream is captured and read to obtain the content of the variables. Applicants' Specification at page 13, lines 27-28, indicates that the content of the variables is added to the data object. The command from the application program is executed to retrieve the at least one variable from the data object for return to the application program. For example, see Applicants' Specification, page 14, lines 7-11 and FIG. 3b, block 440.

In another aspect of the invention, the application program is a first application program. A request from a second application program for at least one variable maintained by the operating system is received. The requested at least one variable from the data object populated

as a result of the command executed by the first application program is returned. For example, Applicants' Specification, page 14, lines 14-19, indicates that environment variables are stored in a data object accessible to te application program that requested the environment variables and may be made available to a separate application program for continues use after a single call to the native operating system command to access the environment variables.

In a further aspect of the invention, the at least one variable retrieved as a result of execution of the command from the application program is a set of environment variables. For example, Applicants' Specification, page 12, lines 13-14, describe that an environmental variable may be obtained when an application program requests the environment variable.

In yet another aspect of the invention, a type of the operating system is determined. For example, see Applicants' Specification at page 13, lines 3. The operating system native command is selected from a set of native operating system commands for different types of operating systems. For example, see Applicants' Specification, page 5, lines 9-10. The selected operating system native command is capable of being executed on the operating system to retrieve the at least one variable, and the application program is capable of executing on each of the different types of operating systems. For example, see Applicants' Specification, page 5, lines 10-13. Also, an application program may access operating system information using one of many available native operating system commands. For example, see Applicants' Specification, page 14, lines 26-28.

In yet a further aspect of the invention, the command from the application program and the operating system native command are executed in a first process and the application program is executed in a second process. See, for example, Applicants' Specification, page 13, lines 7-11.

In another aspect of the invention, the command from the application program is for storing multiple variables and retrieving the variables comprises generating a data stream including the variables. Variables are read from the data stream into the buffer and each line in the buffer is processed to determine each variable name and value, wherein each determined variable name and value is stored in the data object. For example, see Applicants' Specification, page 13, lines 11-17.

In yet another aspect of the invention, determining each variable name and value includes determining a location of an equal sign, setting the variable name to the string preceding the equal sign, and setting the variable value to the string following the equal sign. For example, see Applicants' Specification, page 13, lines 22-27.

In a further aspect of the invention, the variable name and value for each variable are maintained on at least one line. Each line in the data stream is processed line-by-line. It is determined whether each line includes the equal sign, wherein, for each line including the equal sign, the variable name is set to the string preceding the equal sign and the variable value is set to the string following the equal sign. Then, the content of each line not including the equal sign is appended to the variable value. For example, see Applicants' Specification, page 13, line 11 - page 14, line 4.

VI. Issues

A concise statement of the issues presented for review is as follows:

A. Issue 1: The Anticipation Rejection Based on the Petrusha reference

Whether the Examiner is correct in rejecting claims 1, 3, 4, 6, 7, 8, 10, 12, 13, 15, 16, 17,

19, 21, 22, 24, 25, and 26 under 35 U.S.C. 102(b) as being anticipated by Ron Petrusha, "Inside the Windows 95 Registry" (hereinafter "Petrusha").

B. Issue 2: The Obviousness Rejection Based on the Petrusha Reference

Whether the Examiner is correct in rejecting claims 5, 9, 14, 18, 23, and 27 under 35 U.S.C. 103(a) as being unpatentable over Petrusha as applied to claims 1, 8, 10, 17, 19, and 26.

VII. Grouping of the Claims

With respect to Issue 1 (The Anticipation Rejection Based on the Petrusha reference), claims 1, 4, 6, 10, 13, 15, 19, 22, and 24 should be treated as one group (Group 1A), claims 3, 12, and 21 should be treated as a separate group (Group 1B), and claims 7, 8, 16, 17, 25, and 26 should be treated as a separate group (Group 1C).

With respect to Issue 2 (The Obviousness Rejection Based on the Petrusha reference), claims 5, 14, and 23 should be treated as one group (Group 2A), and claims 9, 18, and 27 should be treated as a separate group (Group 2B).

VIII. Argument

A. Issue 1: The Rejection of Claims 1, 3, 4, 6, 7, 8, 10, 12, 13, 15, 16, 17, 19, 21, 22, 24, 25, and 26 as Anticipated by the Petrusha Reference Should be Reversed.

1. Claims 1, 4, 6, 10, 13, 15, 19, 22, and 24 (Group 1A) are not Anticipated by the Petrusha Reference.

As set forth above, claims 1, 10, and 19 (Group 1A) are directed to a method, system, and

article of manufacture for accessing variables from an operating system. A command is received from an application program for at least one variable maintained by the operating system. It is determined whether the at least one variable is in a data object. If the at least one variable is in the data object, returning the at least one variable to the application program, and, if the at least one variable is not in the data object, the command from the application program is executed to store at least one variable maintained by the operating system in the data object accessible to the application program, wherein the application program is executing on the operating system. In particular, an operating system native command to use to retrieve the at least one variable is determined. The operating system native command is executed in response to the command from the application program to retrieve the at least one variable into a buffer. The retrieved at least one variable is stored from the buffer into the data object. The command from the application program is executed to retrieve the at least one variable from the data object for return to the application program.

The law is clear that a claim is anticipated "only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference." Verdegaal Bros. V. Union Oil Co. of California, 2 U.S.P.Q. 2d 1051, 1053 (Fed. Cir. 1987). In that the Examiner has failed to cite any express description or provide support for any inherent description in the Petrusha reference for executing a command from an application program to store at least one variable maintained by the operating system in the data object accessible to the application program, wherein the application program is executing on the operating system, by determining an operating system native command to use to retrieve the at least one variable and executing the operating system native command in response to the command from the

application program to retrieve the at least one variable into a buffer, it is clear that the rejection of claims 1, 10, and 19 as anticipated by the Petrusha reference should be reversed.

The Examiner cites the Petrusha reference on page 35 pages 38-41 as teaching the subject matter of claims 1, 10, and 19. The cited portion of the Petrusha reference describes a registry, which is a database, and a Registry Editor ("RegEdit") that provides a user interface for browsing the registry. Also, at pages 61-68, the Petrusha reference describes that RegEdit depends on the registry functions within the Win32 API to gather information (page 61).

It is the Examiner's position that the claimed elements of determination of an operating system native command to retrieve at least one variable, execution of the operating system native command in response to the command from an application program, and storing the retrieved at least one variable from the buffer into a data object are disclosed by the registry editor interacting with the operating system's registry to retrieve data through the WIN32 Registry API. Applicants traverse. Unlike the technique described in the Petrusha reference, with Applicants' claimed invention, for different operating systems, the same command from an application program will cause different operating system native commands to be executed to retrieve a variable. Thus, with Applicants' claimed invention an operating system native command is determined for the command from the application program. On the other hand, with the Petrusha reference, the same registry function is used to retrieve a variable, and there is no need to determine an operating system native command. Thus, the Petrusha reference does not anticipate the claimed subject matter.

It is the Examiner's position that the Registry Editor is an application program. Applicants traverse and submit that the Registry Editor is an operating system tool. For example,

Applicants are submitting with this appeal brief, as Appendix B, a document printed from http://support.microsoft.com/default.aspx?scid=kb;en-gb;835823, retrieved on March 8, 2004, which indicates that "Windows comes with a tool called the Registry Editor for making changes" (see "How Can I Access the Registry to Make Changes to it?" on first page of document). Because the Windows 95 editor comes with the Registry Editor tool, there is no need in the Petrusha reference to determine an operating system native command. In particular, the registry functions within the Win32 APIs invoked by the Registry Editor are executed without further determining an operating system native command.

Moreover, in the claimed invention, the command from the application program is executed to retrieve the variable from the data object for return to the application program. It is the Examiner's position that executing the command from the application program to retrieve the at least one variable from the data object for return to the application program is anticipated with "displaying the tree structure." However, there is no support for the Examiner's conclusory statement that "displaying the tree structure" anticipates executing the command from the application program to retrieve the at least one variable from the data object for return to the application program.

Also, the Petrusha reference describes that the RegEdit user interface includes a left-hand pane (a "key pane") that displays registry keys and subkeys (page 38). The key pane of the RegEdit user interface lets a user know which registry keys have unexpanded children and which have expanded children (page 64). Moreover, the Petrusha reference describes a ReqQueryInfoKey function that determines the length of a registry key's longest subkey name to prevent allocating too small a buffer for retrieving subkey names, and a RegEnumKeyEx

function that retrieves the name of the first subkey belonging to each top-level key retrieving values (page 65). Although the Petrusha references describes a buffer and data in the buffer being returned to the RedEdit program, the Petrusha reference does not describe storing data from the buffer into a data object as claimed.

It is the Examiner's position that the Petrusha reference's description that if a selected key's subkey information has not yet been gathered by the program, the subkeys are retrieved, enumerated, and added as nodes in the TreeView control (page 68) as disclosing determining whether the requested variable is in the data object, wherein the command from the application program is executed to store at least one variable maintained by the operating system in the data object accessible to the application program. Claims 1, 10, and 19 describe determining whether the at least one variable is in a data object, if the at least one variable is in a data object, returning the at least one variable to the application program, and, if the at least one variable is not in the data object, executing the command from the application program to store at least one variable maintained by the operating system in the data object accessible to the application program, wherein the application program is executing on the operating system. The Petrusha reference does not describe the claimed data object, into which data from a buffer is later stored. Furthermore, the description of Petrusha's reference that if a selected key's subkey information has not yet been gathered by the program, the subkeys are retrieved, enumerated, and added as nodes in the TreeView control does not anticipate returning the at least one variable to the application program if the at least one variable is in a data object, and executing the command from the application program to store at least one variable maintained by the operating system in the data object accessible to the application program if the at least one variable is not in the data

object.

Accordingly, the rejection of claims 1, 10, and 19 should be reversed.

Dependent claims 4, 13, and 22 depend from claims 1, 10, and 19 and further describe that the at least one variable retrieved as a result of execution of the command from the application program is a set of environment variables. Although the Petrusha reference describes environment variables, the Petrusha reference does not describe retrieving at least one variable as disclosed in claims 1, 10, and 19. Accordingly, the rejection of claims 4, 13, and 22 should be reversed.

Dependent claims 6, 15, and 24 depend from claims 1, 10, and 19 and further describe that the command from the application program and the operating system native command are executed in a first process and the application program is executed in a second process. It is the Examiner's position that the registry editor and the WIN32 registry API are inherently separate processes. First, as discussed above, the registry editor is not an application program. Second, regardless of whether the registry editor and the WIN32 registry API are separate processes, the Petrusha reference does not describe, for example, executing a command from an application program to store at least one variable maintained by the operating system in a data object accessible to the application program, wherein the application program is executing on the operating system, by determining an operating system native command to use to retrieve the at least one variable and executing the operating system native command in response to the command from the application program to retrieve the at least one variable into a buffer. Accordingly, the rejection of claims 6, 15, and 24 should be reversed.

Accordingly, it is respectfully submitted that the rejection of claims 1, 4, 7, 14, 17, 20, 27,

30, 33, 40, 43, and 46 (Group 1A) as anticipated by the Petrusha reference should be reversed.

2. Claims 3, 12, and 21 (Group 1B) are not Anticipated by the Petrusha Reference.

Claims 3, 12, and 21 (Group 1B) describe that the application program is a first application program, receiving a request from a second application program for at least one variable maintained by the operating system, and returning the requested at least one variable from the data object populated as a result of the command executed by the first application program.

The Examiner has cited no explicit teaching or suggestion in the Petrusha reference to show that the application program is a first application program, receiving a request from a second application program for at least one variable maintained by the operating system, and returning the requested at least one variable from the data object populated as a result of the command executed by the first application program.

The Examiner cites "Accessing the Registry on Remote Computers" on pages 60-61 of the Petrusha reference as indicating that a server program interacts with the remote computer's registry via a WIN32 registry API on behalf of a client program and that registry entries obtained by the server program are transmitted to the client and displayed within a client-side registry editor interface. Applicants traverse. The Petrusha reference merely provides a technique for viewing the registry of a remote computer.

On the other hand, Applicants' invention receives a request from a second application program for at least one variable maintained by the operating system, and the requested at least one variable is returned *from the data object populated as a result of the command executed by*

the first application program. On the other hand, the Petrusha reference does not provide any indication that a variable is stored in a data object so that it may be returned in response to a command from more than one application program requesting that variable. For example, Applicants' Specification, page 14, lines 14-19, indicates that environment variables are stored in a data object accessible to te application program that requested the environment variables and may be made available to a separate application program for continues use after a single call to the native operating system command to access the environment variables.

Accordingly, it is respectfully submitted that the rejection of claims 3, 12, and 21 (Group 1B) as anticipated by the Petrusha reference should be reversed.

3. Claims 7, 8, 16, 17, 25, and 26 (Group 1C) are not Anticipated by the Petrusha Reference.

Claims 7, 16, and 25 (Group 1C) describe the command from the application program is for storing multiple variables, and wherein retrieving the variables comprises generating a data stream including the variables, which includes reading the variables from the data stream into the buffer and processing each line in the buffer to determine each variable name and value, wherein each determined variable name and value is stored in the data object. Also, the variables are retrieved by executing the operating system native command in response to the command from the application program (see claims 1, 10, and 19 from which claims 7, 16, and 25, respectively, depend).

The Examiner has cited no explicit teaching or suggestion in the Petrusha reference to show that the command from the application program is for storing multiple variables, and

wherein retrieving the variables comprises generating a data stream including the variables, which includes reading the variables from the data stream into the buffer and processing each line in the buffer to determine each variable name and value, wherein each determined variable name and value is stored in the data object.

The Examiner cites the Petrusha reference on pages 67-68 that detail the expansion of a node in a key pane. The Petrusha reference describes that the RegEdit user interface includes a left-hand pane (the "key pane") that displays registry keys and subkeys (page 38). The key pane of the RegEdit user interface lets a user know which registry keys have unexpanded children and which have expanded children (page 64). At pages 67-68, the Petrusha reference describes generation of a key page that displays keys and their children. However, the generation of such a user interface does not anticipate, for example, that the command from the application program is for storing multiple variables, and wherein retrieving the variables comprises generating a data stream including the variables. For example, instead of a command being executed by an application program, pages 67-68 describe code that is part of initialization code (page 66) that appears to be initiated without a command being executed by any application program. Also, the generation of a user interface "key pane" does not describe that variables are retrieved by generating a data stream and that the variables are read from the data stream into a buffer, with each line in the buffer being processed to determine each variable name and value and wherein each determined variable name and value is stored in the data object. It is the Examiner's position that node names are read into buffers, but there is no description that the node names are in a data stream or that variable names and values are stored from the buffers into data objects. Furthermore, it is the Examiner's position that using node information to generate additional

entries describes processing each line in the buffer to determine each variable name and value, wherein each determined variable name and value is stored in the data object. However, there is no explicit teaching or suggestion in the Petrusha reference describing processing each line in a buffer to determine a variable name and value or to store the determined variable name and value in a data object.

Claims 8, 17, and 26 (Group 1C) depend from claims 7, 16, and 25 and further describe that determining each variable name and value when processing each line in a buffer includes determining a location of an equal sign, setting the variable name to the string preceding the equal sign, and setting the variable value to the string following the equal sign.

It is the Examiner's position that the subject matter of claims 8, 17, and 26 are inherently performed because page 43 illustrates a sample .REG file in which value entries for particular keys are stored in a "variable=value" format, where "variable" and "value" are strings, and that by parsing the lines of the file and populating the registry tree data structure, the equal sign is inherently being recognized by the parser. Applicants' traverse. "A claim limitation is inherent in the prior art if it is necessarily present in the prior art, not merely probably or possibly present." Rosco v. Mirror Lite, 64 U.S.P.Q. 2d 1676, 1680 (Fed. Cir. 2002). The Examiner provides no support that determining each variable name and value when processing each line in a buffer, where variables from a data stream have been read into the buffer in response to executing an operating system native command in response to the command from the application program (see claims 1, 10, and 19 from which claims 8, 17, and 26, respectively, depend), is necessary in the Petrusha reference. Additionally, even if the .REG file is parsed, parsing of a file is not equivalent to processing each line in a buffer where variables from a data stream have

been read into the buffer in response to executing an operating system native command in response to the command from the application program.

Accordingly, it is respectfully submitted that the rejection of claims 7, 8, 16, 17, 25, and 26 (Group 1C) as anticipated by the Petrusha reference should be reversed.

B. Issue 2: The Rejection of Claims 5, 9, 14, 18, 23, and 27 as Obvious over the Petrusha Reference Should be Reversed.

1. Claims 5, 14, and 23 (Group 2A) are not Obvious over the Petrusha Reference.

Claims 5, 14, and 23 describe determining a type of the operating system and selecting the operating system native command from a set of native operating system commands for different types of operating systems, wherein the selected operating system native command is capable of being executed on the operating system to retrieve the at least one variable, and wherein the application program is capable of executing on each of the different types of operating systems.

The Examiner has failed to establish that the Petrusha reference has any description, express or inherent, for example, of determining a type of the operating system and selecting the operating system native command from a set of native operating system commands for different types of operating systems.

The Examiner indicates that pages 183-206 discuss various platforms on pages 183-206, but the Petrusha reference in these pages describe one platform, a Windows95 platform, and different programs, Win16 and DOS programs that run under same Windows 95 platform and can access the registry (page 183). For example, DOS programs running under the Windows 95

platform access the registry with parameter validation, by pushing parameters on the stack, and moving function numbers into AX, and calling the VMM (page 191). Thus, the different programs are running on the same Windows 95 platform, so there is no need to select a native operating system command.

Claims 5, 14, and 23 describe determining a type of the operating system. It is the Examiner's position that the last paragraph of page 186 and code examples on page 187 and pages 521-629 teach determining a type of operating system with the expressed motivation of knowing which platforms (Windows 3.1, Windows 95, and Windows NT) a registry-enabled application is running on in order to allow for compensation for differences in the registry APIs and the registries themselves, and the Petrusha reference at these pages describes that since the registries themselves are so different in Windows 3.1, Windows 95, and Windows NT, it is important to know which platform the registry enabled application is running on. Applicants traverse. Although a type of operating system is determined, there is no teaching or suggestion that an operating system native command is selected in response to a command received from an application program. Instead, because each operating system includes a Registry Editor tool, there is no need to determine a type of operating system to execute a Registry Editor function.

Moreover, with reference to claims 5, 14, and 23 the Examiner indicates that the Petrusha patent fails to expressly disclose selecting the operating system native command from a set of native operating system commands for different types of operating systems, wherein the application program is capable of executing on each of the different types of operating systems, but that one of ordinary skill in the art would recognize that because of the amount of backward compatibility built into various Microsoft® Windows® platforms it has been well known to have

applications capable of running on multiple platforms and that it would have been obvious to include the selection of operating system commands to access environment variables (see page 10 of Office Action). Applicants traverse. Because the Registry Editor is a tool that is part of an operating system, each different operating system would include its own version of the Registry Editor. Therefore, there would be no need to determine the operating system native command from a set of native operating system commands for different types of operating systems, wherein the application program is capable of executing on each of the different types of operating systems.

Accordingly, it is respectfully submitted that the rejection of claims 5, 14, and 23 over the Petrusha reference should be reversed.

2. Claims 9, 18, and 27 (Group 2B) are not Obvious over the Petrusha Reference.

Claims 9, 18, and 27 (Group 2B) describe that the variable name and value for each variable are maintained on at least one line, processing each line in the data stream line-by-line, determining whether each line includes the equal sign, wherein, for each line including the equal sign, the variable name is set to the string preceding the equal sign and the variable value is set to the string following the equal sign, and appending the content of each line not including the equal sign to the variable value.

The Examiner takes Official Notice that it has been known to employ line wrapping text files when a line exceeds a predetermined length.

Claims 9, 18, and 27 depend indirectly from independent claims 1, 10, and 19, and by their dependency, claims 9, 18, and 27 incorporate the claim requirement, for example, that an

operating system native command be determined in response to receiving a command from an application program for at least one variable and add additional novel claim requirements, which is not taught or suggested by the Petrusha reference.

Accordingly, it is respectfully submitted that the rejection of claims 9, 18, and 27 over the Petrusha reference should be reversed.

IX Conclusion

Each of the rejections set forth in the final Office Action is improper and should be reversed.

Respectfully submitted,

Janaki K. Davda
Reg. No. 40,684

Dated: May 25, 2003

Direct All Correspondence to:
David Victor
Konrad Raynes Victor & Mann LLP
315 South Beverly Drive, Ste. 210
Beverly Hills, California 90212
Tel: 310-553-7977
Fax: 310-556-7984

X. Appendix A

The claims on appeal are as follows:

1. (Previously Presented) A method for accessing variables from an operating system,

comprising:

receiving a command from an application program for at least one variable maintained by the operating system;

determining whether the at least one variable is in a data object;

if the at least one variable is in the data object, returning the at least one variable to the application program; and

if the at least one variable is not in the data object,

executing the command from the application program to store at least one variable maintained by the operating system in the data object accessible to the application program, wherein the application program is executing on the operating system by:

determining an operating system native command to use to retrieve the at least one variable;

executing the operating system native command in response to the command from the application program to retrieve the at least one variable into a buffer;

storing the retrieved at least one variable from the buffer into the data object; and

executing the command from the application program to retrieve the at least one variable from the data object for return to the application program.

2. (Cancelled)

3. (Original) The method of claim 1, wherein the application program is a first application program, further comprising:

receiving a request from a second application program for at least one variable maintained by the operating system; and

returning the requested at least one variable from the data object populated as a result of the command executed by the first application program.

4. (Previously Presented) The method of claim 1, wherein the at least one variable retrieved as a result of execution of the command from the application program is a set of environment variables.

5. (Previously Presented) The method of claim 1, further comprising:

determining a type of the operating system; and

selecting the operating system native command from a set of native operating system commands for different types of operating systems, wherein the selected operating system native command is capable of being executed on the operating system to retrieve the at least one variable, and wherein the application program is capable of executing on each of the different types of operating systems.

6. (Previously Presented) The method of claim 1, wherein the command from the

application program and the operating system native command are executed in a first process and the application program is executed in a second process.

7. (Previously Presented) The method of claim 1, wherein the command from the application program is for storing multiple variables, and wherein retrieving the variables comprises generating a data stream including the variables, further comprising:

reading the variables from the data stream into the buffer; and

processing each line in the buffer to determine each variable name and value, wherein each determined variable name and value is stored in the data object.

8. (Original) The method of claim 7, wherein determining each variable name and value comprises:

determining a location of an equal sign;

setting the variable name to the string preceding the equal sign; and

setting the variable value to the string following the equal sign.

9. (Previously Presented) The method of claim 8, wherein the variable name and value for each variable are maintained on at least one line, further comprising:

processing each line in the data stream line-by-line;

determining whether each line includes the equal sign, wherein, for each line including the equal sign, the variable name is set to the string preceding the equal sign and the variable value is set to the string following the equal sign; and

appending the content of each line not including the equal sign to the variable value.

10. (Previously Presented) A computer system for accessing variables from an operating system, comprising:

a computer;

a memory storing at least one variable;

program logic executed by the computer, comprising:

means for receiving a command from an application program for at least one variable maintained by the operating system;

means for determining whether the at least one variable is in a data object;

if the at least one variable is in the data object, means for returning the at least one variable to the application program; and

if the at least one variable is not in the data object,

means for executing the command from the application program to store at least one variable maintained by the operating system in the data object in the memory accessible to the application program, wherein the application program is executing on the operating system with:

(i) means for determining an operating system native command to use to retrieve the at least one variable;

(ii) means for executing the operating system native command in response to the command from the application program to retrieve the at least one variable into a buffer;

(iii) means for storing the retrieved at least one variable from the buffer

into the data object; and

(iv) means for executing the command from the application program to retrieve the at least one variable from the data object for return to the application program.

11. (Cancelled)

12. (Original) The system of claim 10, wherein the application program is a first application program, wherein the program logic further comprises:

means for receiving a request from a second application program for at least one variable maintained by the operating system; and

means for returning the requested at least one variable from the data object populated as a result of the command executed by the first application program.

13. (Previously Presented) The system of claim 10, wherein the at least one variable retrieved as a result of execution of the command from the application program is a set of environment variables.

14. (Previously Presented) The system of claim 10, wherein the program logic further comprises:

means for determining a type of the operating system; and

means for selecting the operating system native command from a set of native operating system commands for different types of operating systems, wherein the selected operating system

native command is capable of being executed on the operating system to retrieve the at least one variable, and wherein the application program is capable of executing on each of the different types of operating systems.

15. (Previously Presented) The system of claim 10, wherein the program logic further comprises means for executing the command from the application program and the operating system native command in a first process and mean for executing the application program in a second process.

16. (Previously Presented) The system of claim 10, wherein the command from the application program is for storing multiple variables, and the program logic for retrieving the variables comprises means for generating a data stream including the variables, and where the program logic further comprises:

means for reading the retrieved variables from the data stream into the buffer; and

means for processing each line in the buffer to determine each variable name and value, wherein each determined variable name and value is stored in the data object.

17. (Original) The system of claim 16, wherein the program logic for determining each variable name and value comprises:

means for determining a location of an equal sign;

means for setting the variable name to the string preceding the equal sign; and

means for setting the variable value to the string following the equal sign.

18. (Previously Presented) The system of claim 17, wherein the variable name and value for each variable are maintained on at least one line, and wherein the program logic further comprises:

means for processing each line in the data stream line-by-line;

means for determining whether each line includes the equal sign, wherein, for each line including the equal sign, the variable name is set to the string preceding the equal sign and the variable value is set to the string following the equal sign; and

means for appending the content of each line not including the equal sign to the variable value.

19. (Previously Presented) An article of manufacture for use in accessing variables from an operating system, the article of manufacture comprising computer useable media accessible to a computer, wherein the computer usable media includes at least one computer program that is capable of causing the computer to perform:

receiving a command from an application program for at least one variable maintained by the operating system;

determining whether the requested at least one variable is in a data object;

if the requested at least one variable is in the data object, returning the at least one variable to the application program; and

if the requested at least one variable is not in the data object,

executing the command from the application program to store at least one variable maintained by the operating system in the data object accessible to the application program,

wherein the application program is executing on the operating system by:

determining an operating system native command to use to retrieve the at least one variable;

executing the operating system native command in response to the command from the application program to retrieve the at least one variable into a buffer;

storing the retrieved at least one variable from the buffer into the data object; and

executing the command from the application program to retrieve the at least one variable from the data object for return to the application program.

20. (Cancelled)

21. (Original) The article of manufacture of claim 19, wherein the application program is a first application program, further comprising:

receiving a request from a second application program for at least one variable maintained by the operating system; and

returning the requested at least one variable from the data object populated as a result of the command executed by the first application program.

22. (Previously Presented) The article of manufacture of claim 19, wherein the at least one variable retrieved as a result of execution of the command from the application program is a set of environment variables.

23. (Previously Presented) The article of manufacture of claim 19, further comprising:

determining a type of the operating system; and

selecting the operating system native command from a set of native operating system commands for different types of operating systems, wherein the selected operating system native command is capable of being executed on the operating system to retrieve the at least one variable, and wherein the application program is capable of executing on each of the different types of operating systems.

24. (Previously Presented) The article of manufacture of claim 19, wherein the command from the application program and the operating system native command are executed in a first process and the application program is executed in a second process.

25. (Previously Presented) The article of manufacture of claim 19, wherein the command from the application program is for storing multiple variables, and wherein retrieving the variables comprises generating a data stream including the variables, and further comprising:

reading the retrieved variables from the data stream into the buffer; and

processing each line in the buffer to determine each variable name and value, wherein each determined variable name and value is stored in the data object.

26. (Original) The article of manufacture of claim 25, wherein determining each variable name and value comprises:

determining a location of an equal sign;

setting the variable name to the string preceding the equal sign; and

setting the variable value to the string following the equal sign.

27. (Previously Presented) The article of manufacture of claim 26, wherein the variable name and value for each variable are maintained on at least one line, further comprising:

processing each line in the data stream line-by-line;

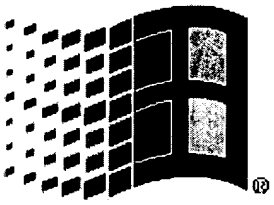
determining whether each line includes the equal sign, wherein, for each line including the equal sign, the variable name is set to the string preceding the equal sign and the variable value is set to the string following the equal sign; and

appending the content of each line not including the equal sign to the variable value.

XI. Appendix B

Microsoft Knowledge Base Article - 835823

Registry problems



The Registry lies at the heart of Windows, controlling how your OS interacts with your hardware and applications. So what do you do when it starts to play up?

GETTING STARTED

The Windows Registry is a database of settings for all the hardware, software and different user preferences on your PC. Whenever you add hardware, uninstall a program or change the resolution of your display, it's the Registry that holds the data about this change.

It works away in the background and is not designed to be accessed or edited by the casual Windows user. This is because making an incorrect entry in the Registry can cause major problems, including your PC no longer starting. For this reason you shouldn't start tweak Registry settings unless you're confident about the changes you're making and have a backup plan in place. That said, there are times when being able to access and edit the Registry can be incredibly handy, healing a sick PC in minutes. We're going to show you how to access and back up your Registry, plus solve common problems with a simple setting change.

HOW CAN I ACCESS THE REGISTRY TO MAKE CHANGES TO IT?

The Registry data is contained within a handful of files, depending on the Operating System you use. For example, in Windows 98 there are two hidden files called User.dat and System.dat in your Windows directory. However, you can't open and edit these files directly and Windows comes with a tool called the Registry Editor for making changes. You can access this tool in all versions of Windows by pressing Start, Run, typing regedit and pressing OK.

HOW DO I GO ABOUT BACKING UP THE REGISTRY?

Understanding whether you have a good backup and how to restore it if things go wrong is one of the most important questions you should address before attempting to make any changes. The process varies according to your version of Windows.

For Windows 98 press Start, Run, type scanregw and click OK. When you receive a prompt to back up the Registry click Yes and then press OK when you're informed that the process is complete.

MY WINDOWS 98 REGISTRY IS WORKING OK, BUT THE ASSOCIATED FILES HAVE GROWN MASSIVELY AND THERE ARE LOADS OF REDUNDANT ENTRIES IN THERE.

The longer you run a PC without a clean reinstall, the more your Registry files grow. This is just a fact of Windows and over time it can cause your PC to slowdown. While Windows is good at adding and tracking changes in your Registry, tasks such as uninstalling software do not always cause the Registry entries to be removed. There's a utility for Windows 98 called RegClean that can help you get rid of all these erroneous entries and speed up your PC in the process. You can download it from http://download.com.com/3000-2094-881470.html.

How do I create new Keys, String values, or edit value data in the Registry Editor tool?

Just browse to the key you're interested in. If you want to create a new subkey for this key, or a new value within this key, then right-click it and select New. You can then choose to create a new key or new value type as required. To edit value data, browse to the value and simply double-click it. If the data can be edited then a box will appear that you can simply type the new data into.

Sometimes when I click on my mouse, it acts as if I have clicked it twice, rather than once.

There is a setting in the Registry that you can switch on to detect accidental double mouse clicks. Browse to HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced, create a new String value called UseDoubleClickTimer and set the value data to 1.

My Windows text is difficult to read, with jagged edges on the letters. I have checked that the fonts are installed correctly. What can I do?

A Registry setting can smooth the fonts displayed on your PC by using anti-aliasing. It's set on by default, so it may be that this has value has been modified, or deleted. Browse to HKEY_CURRENT_USER\Control Panel\Desktop and see if the String FontSmoothing is present. If it's there set the value to 2. If it isn't, recreate it.

I love the thumbnail view in Explorer, but is there a way that I can make the pictures easier to see?

Browse to HKEY_CURRENT_ USER\Software\Microsoft\Windows\CurrentVersion\Explorer. Create or modify two DWORDs called ThumbnailSize and ThumbnailQuality. For ThumbnailSize set the value in pixels, with the default being 96. For ThumbnailQuality set the value as a number which represents the percentage quality between 50 and 100.

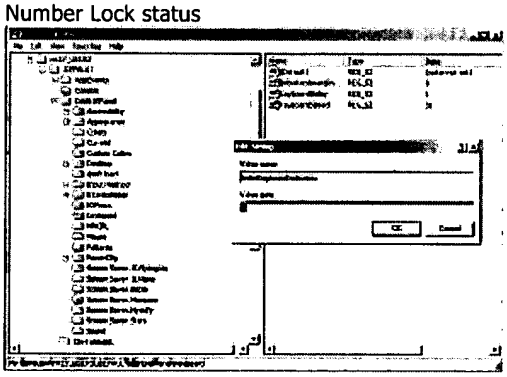
I have programs still listed in the Add/Remove programs dialog even though they have already been uninstalled. How do I get rid of them?

The Registry can help you get rid of these redundant entries. Browse to the following key: HKEY_ LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall.

You will find a collection of subkeys beneath this key, each one representing an entry in the Add/Remove programs dialog. Click any subkey to see the program that it is associated with, which can be viewed in the DisplayName value. Delete the subkey to remove the entry.

A simple Registry tweak

To understand the principles of altering the Registry, follow this simple tweak for changing the Number Lock status



Press Start/Run and type regedit to launch the Registry Editor. Use the Explorer-style browse tree in the left-hand pane to find HKEY_USERS\.DEFAULT\Control Panel\Keyboard.

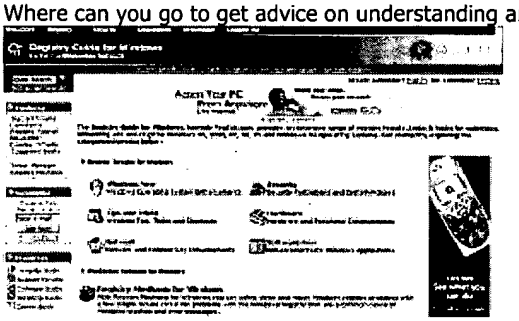
Look in the right-hand pane for the value InitialKeyboardIndicators and double-click it. In the Edit String dialog box change the Value data to either 0 for Off or 2 for On. Press OK and close

the Registry Editor.

The change made will apply to all users, but if you want to make the change just for yourself and not for other users on the PC then you can do this by making the same changes to the following key: HKEY_CURRENT_USER\Control Panel\Keyboard.

REGISTRY HELP

Where can you go to get advice on understanding and modifying the Registry?



If you find yourself in need of help and we're not at hand, there's one online source that stands head and shoulders above the rest.

Formerly known as Regedit.com, the Windows Registry Guide can be found at www.regedit.com. You'll find detailed tutorials explaining what the Registry is and how it works. There are links to a host of utilities that can help you make the most of your Registry and keep it fine tuned. Most important of all, you'll find just about every tweak you'll ever need, divided into helpful sections. These range from fixing problems with hardware and software, through to tips and tricks that will optimise your system.

The Registry is one of the most powerful components of your Operating System and the Windows Registry Guide can help you unlock its full potential. You can even access the contents of the guide when you're not online by downloading and installing a small application from www.winguides.com/guides.php?guide=registry

The information in this article applies to:

- the operating system: Microsoft Windows 98
- the operating system: Microsoft Windows 98 Second Edition
- Microsoft Windows 98 Second Edition

Last Reviewed: 28/01/2004 (1.0)